Business-Oriented Development of Telecommunication Services

Dr Richard Sinnott
GMD Fokus
Kaiserin-Augusta-Allee 31
Berlin, Germany
sinnott@fokus.gmd.de

Mario Kolberg
Dept. Electronic and Electrical Engineering
University of Strathclyde
Glasgow, Scotland
mkolberg@comms.eee.strath.ac.uk

The development of software for distributed systems, e.g. telecommunication services, is a complex activity. Numerous issues have to be resolved when developing such systems, examples of which are language/system heterogeneity and remoteness of components. Interface definition languages (IDLs) are used as the basis for addressing some of these issues. IDLs allow for the specification of the syntactic aspects of the interfaces of the components in the system to be made. Whilst lending itself to issues of heterogeneity and location transparency, dealing with IDL as the basis for system development is not without its problems. Two of the main problems with IDL are its lack of behaviour and its lack of abstraction. Thus designers should not be constrained to work within the syntactic notations used to implement their systems, nor should they be unaided in how they might better design their systems. In this paper we show how these issues are being addressed in the TOSCA project in its development of a service creation and validation environment.

Keywords: Service Creation and Validation, Paradigms, Object-Oriented Frameworks, SDL.

1 Introduction

The basis of IDL in supporting a *message-is-understood* paradigm supports hardware and software heterogeneity (where implementation language mappings on different platforms are provided) as well as remoteness of components issues (where communication is transparently made through ORBs). IDL suffers from two main drawbacks in developing realistic distributed systems however:

- it only supports system interconnectivity and not (the more desireable) system interworking – thus subsystems that understand the messages sent between them will not necessarily interwork correctly, i.e. perform some pre-identified goal.

- it offers too low a level of abstraction from which realistic services are to be developed.

With regard to the first of these points, various works have investigated how IDL can be extended to deal with behavioural issues [X.9041,LarchIDL]. The problem of expressing the *semantics*[1] associated with IDL is, in the most general case, an unachievable one. Whilst specification techniques such as *preconditions, postconditions* and *invariants* can be used to provide guidance to expressing semantics [X.904], such approaches have, for the most part, not been scalable. Or more precisely, these conditions are frequently associated with models of the system that are, to the software developer, e.g. the implementor of the IDL, so disimilar to the software being developed that the question of the applicability of adopting formal notations are often challenged. Put another way, abstraction is a powerful tool for making

[1] i.e. the associated behaviour of the IDL syntactic constructs.

models from which systems may be reasoned about, but if formal techniques are to be more widely accepted then these models should relate more closely to the software itself.

As well as relating more closely to the software itself, the application of formal description techniques should embrace current state of the art techniques in software development. Examples of such techniques include object-orientation [RumEtal] and the *engineering* of software through frameworks [JohnEtal]. Framework based software engineering has arisen to help to realise the holy grail of software engineering: *re-use*. Frameworks are a natural extension of object-oriented techniques. Whilst object technology provides a basis for re-use of code, e.g. through inheritance, it does not provide features to capture the design experience as such. Frameworks have developed to fulfil this need.

A framework can be regarded as a collection of pieces of software or specification fragments that have been developed to produce software of a certain type or niche, e.g. multimedia telecommunication services. A framework is only partially complete. Typically, they are developed so that they have holes or flexibility points in them where specific information is to be inserted. This process is termed *specialisation*. In the case of telecommunication services, this specialisation of the flexibility points is used to develop a multitude of services with slightly differing characteristics [SinnFMDS], e.g. chatline services, conferencing services, news broadcast services, video on demand services etc. Broadly, all of these services allow for connections to be made between users and services which can subsequently be suspended, resumed or terminated. These operations themselves may come from different sources, e.g. conference chairman, the users themselves or inherent features of the service. As well as these core behaviours, other more specific ones are possible, e.g. the sending of invitations to join the service is likely to be a feature of a multimedia conference, but unlikely to be possible from within a chatline service say. There announcements to join the service are more likely to be sent out using other mechanisms, e.g. email. Further, typically all of these services enable some form of accounting to be made which can be used to perform charging and billing activities. This itself has numerous manifestations: split charging, reverse charging, peak rate charging etc.

Whilst frameworks are an aid in developing systems, ideally their usage to develop services should not require an indepth knowledge of technologies used to realise them. In the telecommunications world for example, services are often created by business consultants who understand the user requirements on the desired functionality of the service to be created, as opposed to having a detailed understanding of the technologies used to realise them, e.g. how the low level IDL of the service is implemented. Thus, a *business-oriented* level of abstraction is required upon which services can be created. The rest of the paper focuses on how this is being achieved in the TOSCA project.

2 Framework Creation within TOSCA

The TOSCA project is developing an environment in which telecommunication services can be created and subsequently validated. An approach has been advocated based upon object-oriented frameworks and high-level and intuitive graphical tools – so called *paradigm tools* - that can be used by potentially non-technical people, to specialise those frameworks. Of particular importance is that the created services are to be validated.

The starting point for this work is the development of object-oriented frameworks. Given the focus of the work in TOSCA on telecommunication services, the frameworks being built are focused around the TINA Service Architecture [TinaSA]. This architecture introduces the

underlying concepts and provides information on how telecommunication applications and the components they are built from, have to behave. Central to the Service Architecture is the concept of a *session*. Three sessions are identified:

- **access session**: this represents mechanisms to support access to services (service sessions) that have been subscribed to.
- **service session**: includes the functionality to execute and control and manage sessions, i.e. it allows control of the communication session.
- **communication session**: controls communication and network resources required to establish end to end connections.

Currently, the service session has been the main area upon which frameworks are being developed in TOSCA. The main components of the service session are:

- **Service Factory** (SF) – used to allow users to create new service sessions or join existing ones. Typically, a new service session request results in an service session manager (SSM) and a user session manager (USM) being created. A request to join an existing service session results in a USM being created.
- **User Session Manager** – used to control the users participation in a service session. We note here that the user has a graphical application – so called ssUAP - that is used to interact with the USM. As we shall see the functionality offered by the ssUAP, e.g. the buttons and windows available and how the USM responds to them, determine how the user participates in the service.
- **Service Session Manager** – used to control the global service session behaviour.

The TINA documents [TinaSA,TinaSC] provide IDL and Object Definition Language (ODL) [TinaODL] descriptions along with informal text to describe their behaviour of these components. The frameworks developed within TOSCA are based around these syntactic and informal object descriptions. To facilitate the development of formal models of these frameworks, mappings from ODL/IDL to a formal language is necessary. SDL is one of the few languages for which language mappings from ODL/IDL have been made. The following table summarises some of the main features of the ODL/IDL to SDL mapping implemented in TOSCA [BornEtal].

ODL/IDL Structure	SDL Mapping
Group type	block type
Object type	block type
Interface type	process type
Object Reference	PId
Oneway (asynchronous) Operation	signal prefixed with pCALL_
Operation (synchronous)	signal pair. The first signal is prefixed with pCALL_, the second signal prefixed with pREPLY_ or pRAISE_ (if exception raised)
Exception	signal prefixed with pRAISE_
Basic IDL types, e.g. long, short, char, float,..	syntype
any	not supported
enum	newtype with corresponding literals
typedef	syntype
struct	newtype with corresponding structure
constant	synonym

Table 1: Summary of the ODL/IDL to SDL Mapping

Other mappings have been made from IDL to SDL [Björk], however these are based largely around the remote procedure call concept of SDL. There are several problems with mapping IDL operations to remote procedures. For example, they prohibit the raising of exceptions – an essential feature in realistic distributed systems. Also, the client side of the remote procedure call is blocked until the server side returns.

The mapping to SDL results in client stubs and server skeletons being produced (and stored in SDL packages) which are used to express the behaviour of the service session components. An example of the kind of SDL generated for the server skeletons focusing on the lifecycle aspects of the service session components, i.e. so they may be suspended, resumed and terminated is shown in figure 1.

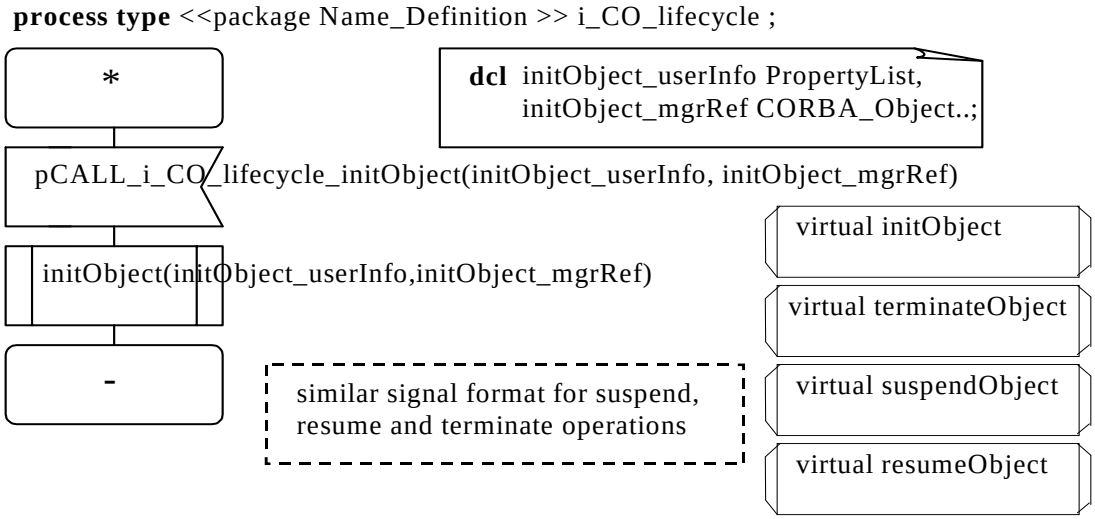

Figure 1: Example of Generated SDL

We note here that signals that result in the operations being called, i.e. the procedures, are accepted in all states. This is the default. These server skeletons may then be filled in, e.g. through inheriting this process type and redefining the behaviour, or through directly inserting the expected functionality in the server skeleton. For this lifecycle process, this entails ensuring that the object can only accept signals to resume or terminate itself when in a suspended state, i.e. after the procedure *suspendObject* has been called successfully.

Specification frameworks are not just prespecified specification fragments however. Instead, they are engineered so that they can be protected as far as possible from specialisations that destroy their integrity, e.g. through the introduction of deadlocks. To accomodate such features, TOSCA has identified certain flexibility points that are available to framework users (and other tools). IDL operations for these flexibility points is given. It is important to note that these flexibility points are not implemented by the framework designer, but by the framework user[2]. The framework designer has to ensure, however, that these flexibility points may only be called at certain times. This in turn is dependent upon the flexibility point. The flexibility points identified thus far have been:

- the start-up, termination, resumption and suspension of user sessions, i.e. the USM component and the objects contained within it. We shall consider how the start up of the user session can be achieved in the next section.

[2] or the paradigm tool used to specialise the framework.

- the start-up, termination, resumption and suspension of service sessions, i.e. the SSM component and the objects contained within it.

- the creation of new user sessions by the SF.

Allowing for this restrictive style of under-specification when developing specifications of frameworks, i.e. so that the specification has *holes* that can only be called at certain times, is quite straightforward to achieve in SDL. The IDL operations representing the flexibility points are mapped to SDL procedures which have null behaviours, i.e. they have a start transition and exit immediately. The actual signal that results in the procedure being called exists in the surrounding interface, i.e. the SDL process. The behaviour of this process can be specified to satisfy whatever constraints are necessary, e.g. the specialisation of the start up of a user session can only occur once the default start up has taken place.

3 Business Oriented Service Development

Whilst frameworks alleviate many of the problems inherent in the development of specifications, to be more directly suited to potentially non-technical service creators, a higher level of abstraction is required. In TOSCA this is achieved through paradigm tools. These tools provide a graphical environment in which services based on frameworks can be developed in an intuitive and more business opriented way. Currently two different paradigms have been considered (and implemented) in TOSCA:

- functional block paradigm which is based on the functional decomposition of the service behaviour. Here the user is provided with a collection (pallette) of building blocks that they may compose together to create services. This composition process defines certain *basic events* that allow for the user to intervene and configure and specialise the building blocks and their combinations.
- movies paradigm where the service is viewed as a series of snapshots which the user defines to reflect the expected service behaviour.

The emphasis on the specification work thus far in TOSCA has been on the functional block paradigm. The identification and IDL specification of a small fixed set of flexibility points readily lends itself to this paradigm based approach since:

- they allow designer intervention at certain key times in the development of the service to tune the behaviour of the service. Thus service creators should not have to be concerned with how low level behaviours are achieved, e.g. how connections are established and subsequently terminated or suspended etc. Rather they should be able to focus on the specific aspects of behaviour that characterise the service being developed.

- they allow for tools to be more easily configured and ported. Thus the issues involved in having different paradigm tools acting on the same framework, or conversely, different frameworks used by the same paradigm tool can be addressed more readily.

- it allows for the paradigm tool SDL code generation process to be simplified. Thus the specialisation process, i.e. filling in the flexibility points, has clearly defined boundaries which correspond to the fixed set of IDL operations.

- they allow the majority of the implementation and specification details to be hidden. Thus the service creator need never be confronted with SDL, the implementation language

used, the ORB calls underlying the communication between the components or even the associated ODL and IDL. Instead they are offered an environment where the service behaviour and those aspects of it that can be influenced, is more meaningfully and intuitively represented.

As we shall see in the next section, this last bullet point applies not only to the development environment, i.e. when using the paradigm tool to create the service from the framework, but also when the behaviour of the created service is checked to ensure it has the expected functionality.

As an example of how the underlying specialisation process is achieved by the paradigm tool, we consider the start up of user sessions as might arise in a multimedia conferencing service. As stated users interact with USMs through a graphical user application, ssUAP. The functionality offered by the ssUAP and how the associated callbacks are dealt with by the USM, can be modified directly. We assume that the default behaviour of the ssUAP[3] is that the user can terminate or suspend their participation in the service.

As a simple specialisation, we assume that the chairman of the conference has the ability to invite other users to join the session or suspend other users already existing in the session. The IDL for the flexibility point of concern is *oneway ufsstart()* which is associated with the manager of the USM component. This is mapped to a virtual procedure that has a start transition followed by an immediate exit. When specialised, this procedure has to add the buttons on the user application (using dynamic widget signals not given here) and extend the USM with features for dealing with the button callback events, e.g. create objects in the USM to handle the behaviour that occurs when the button is pushed.

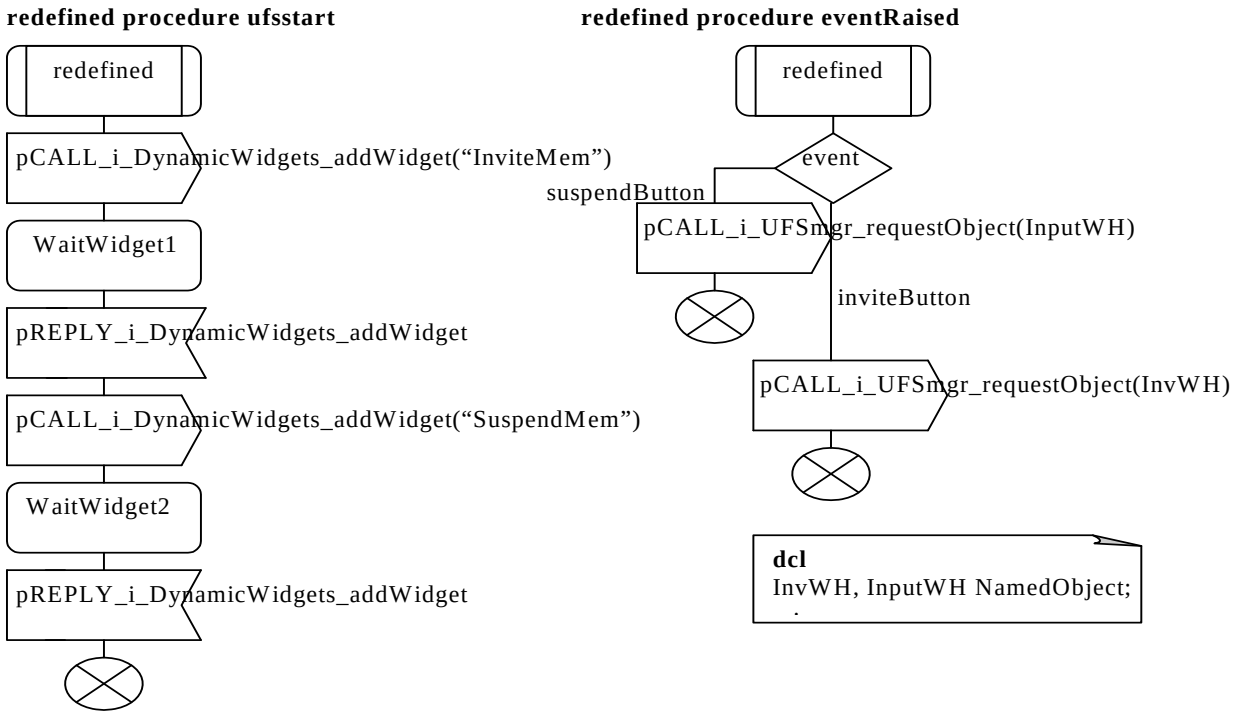

Figure 2: Example of the Specialisation Code Produced

[3] and hence what the USM expects of the ssUAP.

4 Understanding the Behaviour of the Service

Once the design of the service is complete, it is necessary to provide some immediate feedback to the service designer (and potentially the service customer) to ensure that the service behaviour is as desired. Given that the service creator may not necessarily understand the language the service has been created in, e.g. C++ or SDL, an approach based on graphically animating the service behaviour has been adopted.

As well as the animation of the user interface, e.g. new buttons being created on the user application, graphical animations of the interface to the communication session have been made. These show the connections between users in the session and how they are modified when new users join, or existing users suspend or resume their participation. In addition a model of the charges being incurred by users in the session has also been made. It is important to note that the interfaces used in the animation, i.e. the GUIs, are themselves CORBA objects. Currently, C++ wrappers are used to tie the C code generated by the SDL tools (namely the SDL simulator SDT [TAU]) to the CORBA world, i.e. the GUIs.

As explained in the previous section, the user application (ssUAP) is one aspect which can be manipulated through specialisation of the framework by the paradigm tool. Figures 3 and 4 show two examples of specialised Control Windows, namely for a chairman and participant in a videoconference service.

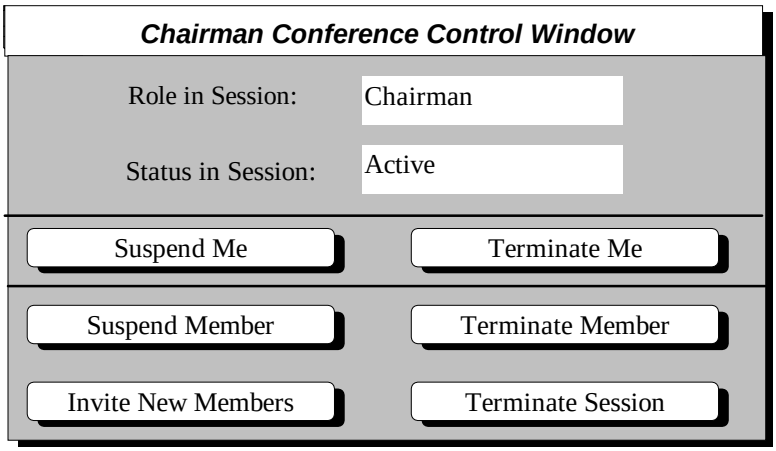

Figure 3: Example of a Control Window for the Chairman of a Conference Session.

The chairman's Control Window extends the default behaviour of suspending and terminating the user participation in the session, to invite new members, terminate or suspend existing session members or terminate the whole session with all members then being terminated.

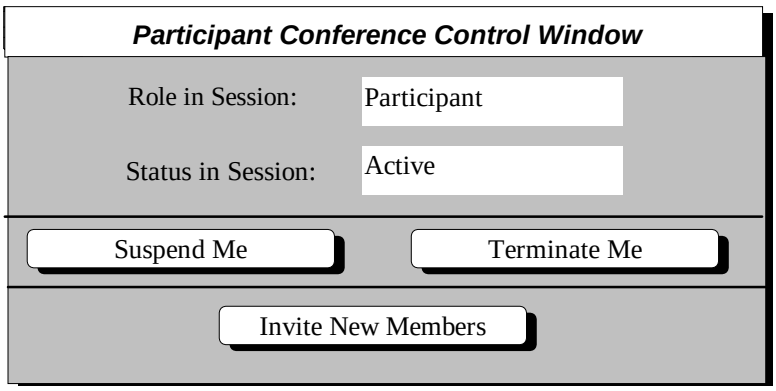

Figure 4: Example of a Control Window for a Participant in a Conference Session.

The Control Window for a participant, however, only allows for their suspension or termination, and the sending of invitations. By using these windows and their associated buttons, the behaviour of the service can be studied and checked. One of the checks made is focused on the connection graph of the TINA communication session, i.e. where stream connections are set-up and their current status. To show this, a connection graph GUI has been developed which connects to the service control logic and contains the same interface as the objects in the communication session, in other words, it emulates the connections being made. An example is depicted in figure 4.

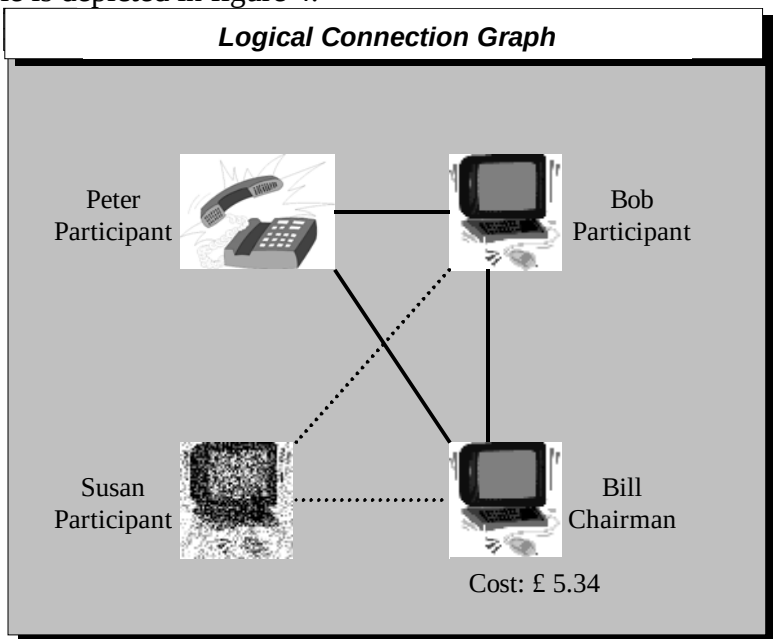

Figure 4: Graphical Interface for showing the connection graph.

Here, there are four members in a conference shown. As the different symbols suggest, three of them (Bob, Bill and Susan) are able to use audio as well as video streams, whereas Peter can only use audio streams. Further, Susan's participation in the service is suspended and hence all her stream connections as well (dashed lines). Also, Bill as the chairman is billed for the conference and hence the current cost is shown attached to him.

This kind of behaviour check can clearly be used by non-technical people (e.g. in a meeting between a business consultant and a potential costumer) to check whether a service meets its expected behaviour. Other GUIs have also been developed to investigate the service behaviour, e.g. invitation GUIs to check that invites are delivered and responded to correctly and within any time constraints. Once the animated behaviour is satisfactory to the service creator, the service can be validated using existing SDL tools [TAU].

5 Conclusions

The idea of engineering specifications and software more generally, is not new. Various works have proposed how specifications might be architected [X.904,X.9041,SinnPhD]. The Architectural Semantics of ODP [X.904,X.9041] in particular has considered how this might be achieved. The primary problem with such works is one of prescriptivity. ODP is an open architecture that does not - and cannot! - prescribe the structure and content of services that might be developed from it. The TINA architecture is more prescriptive since the service kinds that can be created from it are more clearly defined (constrained), i.e. it is known that

they are telecommunication services. Indeed, TINA provides IDL and ODL as a basis for developing these services. We have seen how the tools and techniques used in TOSCA have taken this as the starting point from which SDL specifications of object-oriented frameworks were created and subsequently used to develop models of services.

We argue that this approach is a more realistic and scalable approach to including some form of semantics in the architecture of services, e.g. as opposed to attempting to extend IDL with some form of behaviour model say. Simplistic IDL operations can result in behaviours so complex that it would be neither useful nor practical to attempt to address the semantics of the operation as an extension to the signature say.

Through a framework based approach we have shown that the idea of architecting specifications is no longer a nice idea, but a realistic and applied technique. The exploitation of the framework based approach advocated requires that they are, amongst other things, themselves well designed and easy to use. With regard to the first point, the inherent features of the SDL language, e.g. its support for object-orientation, and the nature of the mapping rules from ODL/IDL to SDL, e.g. in dissociating the behaviour of a flexibility point from its call, make framework development and validation a relatively straightforward activity. The second point has been addressed through the creation of high-level and intuitive graphical user tools in TOSCA. These tools have been engineered to more readily capture user requirements on the service functionality, thus catering directly for a business driven approach to service creation.

The work on the TOSCA project is still ongoing, however initial results in the development of frameworks and paradigm tools have been promising. So far, initial C++ and SDL frameworks have been developed and two paradigm tools implemented. These are currently undergoing first user trials. The emphasis so far in the work has been on the simulation and animation of the services. The next phase of our work will focus more on service validation. This validation activity will consider both the validation of isolated services, and the implementation of services interaction management techniques to support the interworking of services in an environment where other services exist that might adversely influence one another. Some of the issues associated with service interaction in a TINA world are discussed in more detail in [Kolberg].

Another area of TOSCA work is investigating how the code generation of the SDL tools can reflect the SDL system more closely. Thus rather than generate a large C file for the whole system, collections of files are generated that reflect the SDL system structure more closely, e.g. files for the blocks (SDL models of CORBA objects). If successful, this would then allow the generation of CORBA object implementations directly from their SDL models.

More information regarding the TOSCA work can be found at: http://www.teltec.dcu.ie/tosca/

5.1 Acknowledgements

The authors are indebted to the partners in the TOSCA project. The TOSCA consortium consists of Teltec DCU, Silicon & Software Systems Ltd, British Telecommunications, University of Strathclyde, Centro Studi e Laboratori di Telecommunicazioni SpA, Telelogic, Lund Institute of Technology, GMD and Ericsson.

6 References

[Björk] M. Björkander, Mapping IDL to SDL, Telelogic AB, 1997.

[BornEtal] M. Born, A. Hoffmann, M. Winkler, J. Fischer, N. Fischbeck, *Towards a Behavioural Description of ODL*, Proceedings of TINA 97 Conference, Chile.

[JohnEtal] R.Johnson and V.Russo, *Reusing Object-Oriented Designs,*Urbana, Ill., May 1991.

[Kolberg] M. Kolberg and E. Magill: *Service and Feature Interactions in TINA,* accepted for Feature Interaction Workshop'98, Lund, Sweden 1998.

[LarchIDL] G.S. Sivaprasad, *Larch/CORBA:Specifying the Behaviour of CORBA-IDL Interfaces*, Department of Computer Science, Iowa State University, TR 95-27, 1995.

[RumEtal] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modelling and Design*, Prentice Hall 1991.

[SDL92] International Consultative Committee on Telegraphy and Telephony - *SDL - Specification and Description Language*, CCITT Z.100, International Telecommunications Union, Geneva, Switzerland, 1992.

[SinnCSI] R. Sinnott, *Frameworks: The Future of Formal Software Development*, to appear in Semantics of Specifications, Journal of Computer Standards and Interfaces Journal, editor Haim Kilov, August 1998.

[SinnFMDS] R. Sinnott, M. Kolberg, *Engineering Telecommunication Services with SDL*, submitted to International Conference of Formal Methods for Open Object-Based Distributed Systems, Florence Italy, February 1999.

[SinnPhD] R. Sinnott, *An Architecture Based Approach to Specifying Distributed Systems in LOTOS and Z*, PhD Thesis, University of Stirling, Scotland, June 1997.

[TAU] Telelogic AB, *Getting Started Part 1 - Tutorials on SDT Tools*, Telelogic AB, 1997.

[TinaSA] TINA-C, *Service Architecture*, version 5.0, 16 June 1997.

[TinaSC] TINA-C, *Service Component Specification Computational Model and Dynamics*, Version 1.0b (draft 0.1), September 8, 1997.

[TinaODL] TINA-C, TINA Object Definition Language MANUAL, version 2.3, July 1996.

[X.904] *Basic Reference Model of ODP - Part 4: Architectural Semantics*, ISO/IEC International Standard 10746-4, ITU-T Recommendation X.904, Geneva, Switzerland 1997.

[X.9041] *Basic Reference Model of ODP - Part 4: Architectural Semantics Amendment*, ISO/IEC ITU-T, working document ISO/IEC/SC33 *– number to be assigned.*

[YSCE] For more information see http://www.fokus.gmd.de/minos/y.sce